FPGA-based decoder for a Delta-Sigma modulator

TN149 | Posted on August 24, 2021 | Updated on May 7, 2025

Shu WANG
Development Engineer
imperix •

Table of Contents

This technical note shows how to build a decoder IP for a Delta-Sigma Modulator and establish communication with such a device through USR ports of the B-Box RCP and B-Board PRO. The corresponding approach uses the user-programmable area inside the FPGA, also known as *sandbox*.

Introduction

Delta-Sigma Modulators are a class of analog-to-digital converters (ADCs) that produce a high-frequency data stream (1-bit), whose pulse density represents the acquired analog value. In data acquisition applications, such devices are particularly useful when only a few (typ. 1-2) digital lines are available. This may notably be essential when data must be carried across a galvanic isolation barrier, such as in numerous power electronic applications.

Delta-sigma modulation may represent various modulation techniques, resulting in different types of data encoding methods for the digital stream. Common types are Non-Return-to-Zero (NRZ) and Manchester coding. These techniques differ in their bitrates, but may also offer (or not) the possibility of recovering the clock from the bit stream.

Clock recovery is often an essential feature. Indeed, by recovering the data clock directly from the stream itself, a separate clock becomes dispensable, which further reduces the number of required communication lines. In practice, a delta-sigma modulator can communicate with its associated demodulator with **only one digital line**!

This note provides an implementation example centered around the AMC1035 delta-sigma modulator, which supports two output encoding methods: Non-Return-to-Zero (NRZ) and Manchester coding. The data rate of NRZ coding is 9~21MHz while the data rate of Manchester coding is 9~11MHz. In the next chapter, a decoder for each coding method will be provided.

Related notes

- Instructions on how to build an FPGA project template and how to exchange data between the CPU and the sandbox using the AXI4-Stream interface module can be found in Getting started with FPGA control development.
- The page AXI4-Stream IP from Xilinx presents the AXI4-Stream interface and the Xilinx AXI4-Stream IPs.
- Another example of how to expand the number of ADC inputs using SPI and user ports is available in FPGA-based SPI communication IP for ADC.

Software sources

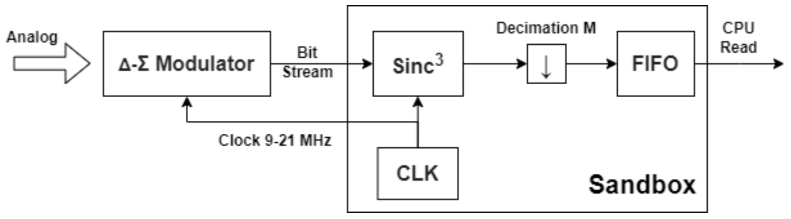
AMC1035Download

Delta-sigma decoder implementation

Synchronous decoder for NRZ encoding

In synchronous mode, the clock signal is generated by the FPGA, transmitted to the delta-sigma modulator, and used by the latter for the modulation. The same clock is then used for decoding the NRZ bit stream that is received by the FPGA. This approach uses two physical USR ports available from the *sandbox* and one three-order sinc filter as a decimator.

In general, the suggested system diagram for a synchronous decoder is shown below.



System diagram using a synchronous decoder

The VHDL codes are given below.

AMC1035 driver using synchronous clock

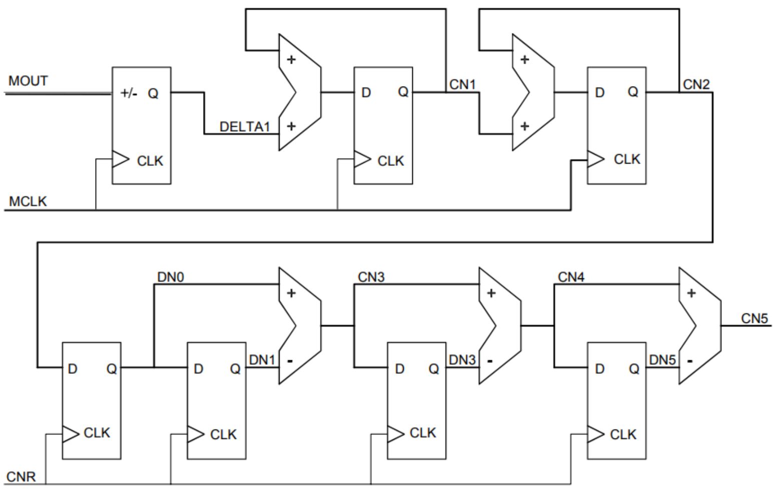
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity amc_driver is
        port(
        -- Configuration from CPU
        AMC_M : in std_logic_vector(15 downto 0); -- Decimation ratio

        -- Input from AMC1035:
        DATA_IN : in std_logic;  -- Input bit stream
        CLK  : in std_logic; -- AMC clock input

        -- sinc3 filter output using AXI4-Stream (Non-blocking)
        M_AXIS_DATA_tdata : out std_logic_vector(31 downto 0);
        M_AXIS_DATA_tvalid : out std_logic;

        -- Active low reset
        RESN : in std_logic
        );
end amc_driver;

architecture rtl of amc_driver is

    ATTRIBUTE X_INTERFACE_INFO : STRING;
    ATTRIBUTE X_INTERFACE_INFO of CLK: SIGNAL is "xilinx.com:signal:clock:1.0 clk CLK";

    -- Divided CLK : f_CNR = f_CLK/M
    signal CNR : std_logic := '0';

    -- Intermediate signals of sinc3 filter
    signal DN0, DN1, DN3, DN5 : std_logic_vector(31 downto 0);
    signal CN1, CN2, CN3, CN4, CN5 : std_logic_vector(31 downto 0);
    signal DELTA1 : std_logic_vector(31 downto 0);

begin

    -- Generate divided CLK
    P_CNR : process(CLK)
        variable CNR_cnt: unsigned(15 downto 0):=(others=>'0');
    begin
        if rising_edge(CLK) then
           M_AXIS_DATA_tvalid <= '0'; -- Default
           -- Toggle CNR
           if CNR_cnt+1 >= unsigned('0' & AMC_M(15 downto 1)) then
               CNR_cnt := (others => '0');
               CNR <= not CNR;
               if CNR = '0' then
                   M_AXIS_DATA_tvalid <= '1'; -- Data is valid at each CNR rising edge
               end if;
           else
```

```vhdl
                CNR_cnt := CNR_cnt + 1;
            end if;
         end if;
     end process P_CNR;

     -- sinc3 filter input
     process(CLK, RESN)
      begin
          if RESN = '0' then
             DELTA1 <= (others => '0');
          elsif CLK'event and CLK = '1' then
             if DATA_IN = '1' then
                 DELTA1 <= DELTA1 + 1;
             end if;
          end if;
      end process;

      -- Integral
      process(RESN, CLK)
      begin
          if RESN = '0' then
             CN1 <= (others => '0');
             CN2 <= (others => '0');
          elsif CLK'event and CLK = '1' then
             CN1 <= CN1 + DELTA1;
             CN2 <= CN2 + CN1;
          end if;
      end process;

      -- Comb
      process(RESN, CNR)
      begin
         if RESN = '0' then
             DN0 <= (others => '0');
             DN1 <= (others => '0');
             DN3 <= (others => '0');
             DN5 <= (others => '0');
         elsif CNR'event and CNR = '1' then
             DN0 <= CN2;
             DN1 <= DN0;
             DN3 <= CN3;
             DN5 <= CN4;
         end if;
      end process;

      CN3 <= DN0 - DN1;
      CN4 <= CN3 - DN3;
      CN5 <= CN4 - DN5;
      M_AXIS_DATA_tdata <= CN5;

end rtl;
```

Code language: VHDL (vhdl)

- The AMC1035 sends a NRZ (Non-Return-to-Zero) coded bit stream
- The clock is generated by the FPGA and fed to the AMC1035 and decoder block, synchronously
- The decimation ratio M can be configured from the CPU using an SBO register
- The output data is a 32-bit unsigned integer available on an AXI4-Stream interface

The sinc^3 filter is implemented using CIC (Cascaded integrator–comb filter) architecture as shown below. CIC is an efficient implementation of a moving-average filter, which is built using adders and registers only. The relationship between the decimation ratio M and the output data width is given in the table below.



Xilinx sinc^3 filter implementation (taken from TI document)

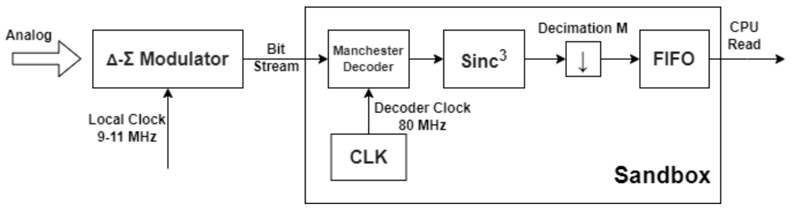
Decimation	Date Rate (kHz)	Gain$_{DC}$ (bits)	Total Output Width (bits)
4	2500	6	7
8	1250	9	10
16	625	12	13
32	312.5	15	16
64	156.2	18	19

Summary of the sin^3 filter for 10MHz samping clock

Manchester decoder

In Manchester coding mode, the AMC1035 can be clocked with a local clock, whose phase and frequency are unknown. On the FPGA side, this approach needs only one user port, for receiving the data stream, and one three-order sinc filter as decimator. In addition, a Manchester decoder is needed to translate Manchester-encoded data to NRZ. This mode has the advantage of reducing the number of used user ports but the drawback of a reduced data rate (down to 9~11MHz).

In general, the suggested system diagram using the Manchester decoder is shown below.



System diagram using Manchester decoder

The VHDL codes are given below.

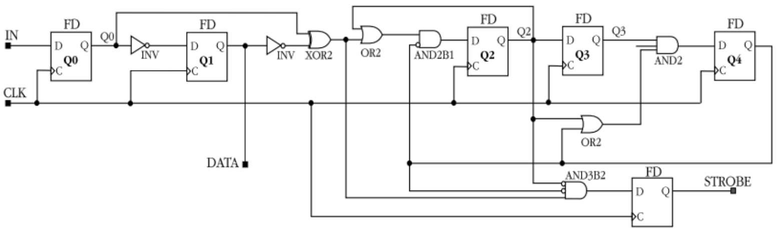
AMC1035 driver using Manchester decoding

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity amc_driver_md is
        port(
        -- Configuration from CPU
        AMC_M: in std_logic_vector(15 downto 0); -- Decimation ratio

        -- Input data from delta-sigma modulator
        DATA_IN  : in std_logic;  -- AMC DATA input

        -- sinc3 filter output using AXI4-Stream (Non-blocking)
        M_AXIS_DATA_tdata : out std_logic_vector(31 downto 0);
        M_AXIS_DATA_tvalid : out std_logic;
```

```vhdl
        -- Decoder clock
        CLK : in std_logic;

        -- Active low reset
        RESN : in std_logic
        );
end amc_driver_md;

architecture rtl of amc_driver_md is

    ATTRIBUTE X_INTERFACE_INFO : STRING;
    ATTRIBUTE X_INTERFACE_INFO of CLK: SIGNAL is "xilinx.com:signal:clock:1.0 clk CLK";

    -- Intermediate signals of Manchester decoder
    signal Q0, Q1, Q2, Q3, Q4 : std_logic := '0';
    signal INV_1, INV_2, XOR2, OR2_1, AND2B1, AND2, OR2_2, AND3B2 : std_logic := '0';

    -- Manchester decoder output
    signal DATA_MD : std_logic; -- Output data
    signal STROBE : std_logic;  -- Output data valid

    -- Divided AMC_CLK : f_CNR = f_CLK/M
    signal CNR_rising_edge : std_logic;
    signal CNR_cnt : unsigned(15 downto 0) := (others=>'0');

    -- Intermediate signals of sinc3 filter
    signal DN0, DN1, DN3, DN5 : std_logic_vector(31 downto 0);
    signal CN1, CN2, CN3, CN4, CN5 : std_logic_vector(31 downto 0);
    signal DELTA1 : std_logic_vector(31 downto 0);

begin

    -- Manchester decoder
    DATA_MD <= INV_2; -- 0: falling 1: rising
    INV_1 <= not Q0;
    INV_2 <= not Q1;
    XOR2 <= Q0 xor INV_2;
    OR2_1 <= XOR2 or Q2;
    AND2B1 <= OR2_1 and (not Q4);
    AND2 <= Q3 and OR2_2;
    OR2_2 <= Q2 or Q4;
    AND3B2 <= (not Q2) and (not Q4) and XOR2;

    P_MD : process(CLK)
    begin
        if rising_edge(CLK) then
            Q0 <= DATA_IN;
            Q1 <= INV_1;
            Q2 <= AND2B1;
            Q3 <= Q2;
            Q4 <= AND2;
            STROBE <= AND3B2;
        end if;
    end process P_MD;

    -- Generate CNR
     P_CNR : process(CLK, RESN)
     begin
         if RESN = '0' then
             CNR_cnt <= (others=>'0');
         elsif rising_edge(CLK) then
             CNR_rising_edge <= '0';
             if STROBE = '1' then -- Data is valid
                 -- Toggle postscaled_CNR
                 if CNR_cnt+1 >= unsigned(AMC_M) then
                     CNR_rising_edge <= '1';
                     CNR_cnt <= (others => '0');
                 else
                     CNR_cnt <= CNR_cnt + 1;
                 end if;
             end if;
         end if;
     end process P_CNR;

     -- sinc3 filter
     P_SINC3 : process(CLK, RESN)
     begin
         if RESN = '0' then
             DELTA1 <= (others => '0');
             CN1 <= (others => '0');
             CN2 <= (others => '0');
             DN0 <= (others => '0');
             DN1 <= (others => '0');
             DN3 <= (others => '0');
             DN5 <= (others => '0');
         elsif rising_edge(CLK) then
             M_AXIS_DATA_tvalid <= '0'; -- Default
             -- Integral
             if STROBE = '1' then -- Data is valid
                 if DATA_MD = '1' then
                     DELTA1 <= DELTA1 + 1;
                 end if;
                 CN1 <= CN1 + DELTA1;
                 CN2 <= CN2 + CN1;
             end if;
             -- Comb
             if CNR_rising_edge = '1' then
                 M_AXIS_DATA_tvalid <= '1';
                 DN0 <= CN2;
                 DN1 <= DN0;
                 DN3 <= CN3;
                 DN5 <= CN4;
             end if;
         end if;
     end process P_SINC3;

     CN3 <= DN0 - DN1;
     CN4 <= CN3 - DN3;
     CN5 <= CN4 - DN5;
     M_AXIS_DATA_tdata <= CN5;

end rtl;
```
Code language: VHDL (vhdl)

- The AMC1035 works in the Manchester coding mode
- The clock used by the Manchester decoder can be asynchronous to the input data, but must be between 5 and 12 times, nominally 8 times, faster than the input data rate. For the AMC1035 we can simply use a 80MHz clock
- The decimation ratio M can be configured by the CPU using an SBO register
- The output data is a 32-bit unsigned integer using AXI4-Stream interface

This FPGA implementation of a Manchester decoder uses two registers and a XOR gate for transition detect, and one divide-by-six Johnson counter that locks up in the 000 state. Once a transition is detected, the `STROBE` flag will be asserted, indicating valid data, then the 6-counter will terminate `STROBE` for the following 5 periods. This procedure ensures that no between-bit transition is detected by mistake.



Manchester decoder circuit (taken from Manchester decoder in 3 CLBs)

The implementation of sinc^3 filter uses the previously introduced CIC architecture. However, due to the different data-valid mechanism, the sinc^3 filter only accepts input data when `STROBE` is asserted, whereas in the previous design, it accepts data at each rising edge of the synchronous clock.

Both blocks have the same name `CLK` for the clock input. They shall however NOT be confused.
For the synchronous decoder block, the `CLK` is the clock used by the AMC1035 running at 9~21MHz.
For the Manchester decoder block, the CLK is a specialized clock for Manchester decoding, running at 80MHz.
The reason why the same name is used for both clocks is that the AXI4-Stream interface is used to simplify the connection between blocks. And any clock used by the AXI4-Stream must be named according to Xilinx conventions. Otherwise, a clock cannot be automatically recognized.

Delta-sigma modulator testbench

For each block, a VHDL testbench simulating NRZ/Manchester coded bit streams is provided to validate the behavior of the two decoders.

Testbench for AMC1035 driver using synchronous clock

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity AMC_driver_tb is
end entity AMC_driver_tb;

architecture rtl of AMC_driver_tb is

constant MCLK_PERIOD : time := 100ns; -- DSM CLK
constant MCLK_LOW    : time := MCLK_PERIOD / 2;
constant MCLK_HIGH   : time := MCLK_PERIOD / 2;
signal mclk : std_logic;

signal RESN : std_logic;

signal DATA_IN : std_logic;
signal M : std_logic_vector(15 downto 0) := std_logic_vector(to_unsigned(8, 16)); -- Decimation ratio

component amc_driver is
        port(
        -- Configuration from CPU
        AMC_M : in std_logic_vector(15 downto 0); -- Decimation ratio

        -- Input from AMC1035:
        DATA_IN : in std_logic;  -- Input bit stream
        CLK  : in std_logic; -- AMC clock input

        -- sinc3 filter output using AXI4-Stream (Non-blocking)
        M_AXIS_DATA_tdata : out std_logic_vector(31 downto 0);
        M_AXIS_DATA_tvalid : out std_logic;

        -- Active low reset
        RESN : in std_logic
        );
end component amc_driver;

-- input NRZ coded '0'
procedure input_0 (signal DATA_IN : inout std_logic) is
begin
    wait until rising_edge(mclk);
    DATA_IN <= '0';
    wait for MCLK_PERIOD;
end procedure input_0;

-- input NRZ coded '1'
procedure input_1 (signal DATA_IN : inout std_logic) is
begin
    wait until rising_edge(mclk);
    DATA_IN <= '1';
    wait for MCLK_PERIOD;
end procedure input_1;

begin
    dut: amc_driver
    port map (
        AMC_M => M,
        DATA_IN => DATA_IN,
        CLK  => mclk,
        M_AXIS_DATA_tdata => open,
        M_AXIS_DATA_tvalid => open,
        RESN => RESN
    );

    -- Reset process
    p_reset : process is
    begin
        wait until falling_edge(mclk);
        RESN <= '0';
        wait for 3*MCLK_PERIOD;
        RESN <= '1';
        wait;
    end process p_reset;

    -- MCLOCK process
    p_mclk: process is
    begin
        mclk <= '0';
```

```vhdl
        wait for 0.4 * MCLK_LOW;
        mclk <= '1';
        WAIT FOR MCLK_HIGH;
        mclk <= '0';
        wait for 0.6 * MCLK_LOW;
    end process p_mclk;

    -- Test process
    p_test : process is
    begin

        -- input "1001"
        input_1(DATA_IN);
        input_0(DATA_IN);
        input_0(DATA_IN);
        input_1(DATA_IN);

--        -- input "1000"
--        input_1(DATA_IN);
--        input_0(DATA_IN);
--        input_0(DATA_IN);
--        input_0(DATA_IN);

--        -- input "1110"
--        input_1(DATA_IN);
--        input_1(DATA_IN);
--        input_1(DATA_IN);
--        input_0(DATA_IN);

        -- Add more input patterns

    end process p_test;

end architecture rtl;
```
Code language: VHDL (vhdl)

Testbench for AMC1035 driver using Manchester decoding

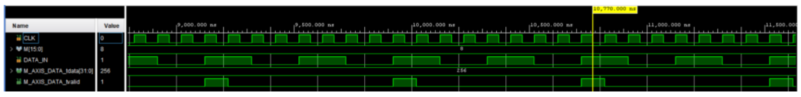
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity AMC_driver_md_tb is
end entity AMC_driver_md_tb;

architecture rtl of AMC_driver_md_tb is

constant CLK_PERIOD : time := 12.5ns; -- Decoder CLK
constant CLK_LOW    : time := CLK_PERIOD / 2;
constant CLK_HIGH   : time := CLK_PERIOD / 2;
signal clk : std_logic;

constant MCLK_PERIOD : time := 100ns; -- DSM CLK
constant MCLK_LOW    : time := MCLK_PERIOD / 2;
constant MCLK_HIGH   : time := MCLK_PERIOD / 2;
signal mclk : std_logic;

signal RESN : std_logic;

signal DATA_IN : std_logic;
signal M : std_logic_vector(15 downto 0) := std_logic_vector(to_unsigned(8, 16)); -- Decimation ratio

component amc_driver_md is
        port(
        -- Configuration from CPU
        AMC_M: in std_logic_vector(15 downto 0); -- Decimation ratio

        -- Input data from delta-sigma modulator
        DATA_IN  : in std_logic;  -- AMC DATA input

        -- sinc3 filter output using AXI4-Stream (Non-blocking)
        M_AXIS_DATA_tdata : out std_logic_vector(31 downto 0);
        M_AXIS_DATA_tvalid : out std_logic;

        -- Decoder clock
        CLK : in std_logic;

        -- Active low reset
        RESN : in std_logic
        );
end component amc_driver_md;
```

```
-- input Manchester coded '0'
procedure input_0 (signal DATA_IN : inout std_logic) is
begin
    wait until rising_edge(mclk);
    DATA_IN <= '1';
    wait until falling_edge(mclk);
    DATA_IN <= '0';
end procedure input_0;

-- input Manchester coded '1'
procedure input_1 (signal DATA_IN : inout std_logic) is
begin
    wait until rising_edge(mclk);
    DATA_IN <= '0';
    wait until falling_edge(mclk);
    DATA_IN <= '1';
end procedure input_1;

begin
    dut: amc_driver_md
    port map (
        AMC_M => M,
        DATA_IN => DATA_IN,
        M_AXIS_DATA_tdata => open,
        M_AXIS_DATA_tvalid => open,
                CLK => clk,
                RESN => RESN
    );

    -- Reset process
    p_reset : process is
    begin
        wait until falling_edge(clk);
        RESN <= '0';
        wait for 3*CLK_PERIOD;
        RESN <= '1';
        wait;
    end process p_reset;

    -- Decoder CLOCK process
    p_clk: process is
    begin
        clk <= '0';
        wait for CLK_LOW;
        clk <= '1';
        WAIT FOR CLK_HIGH;
    end process p_clk;

    -- DSM CLOCK process
    p_mclk: process is
    begin
        mclk <= '0';
        wait for 0.4 * MCLK_LOW;
        mclk <= '1';
        WAIT FOR MCLK_HIGH;
        mclk <= '0';
        wait for 0.6 * MCLK_LOW;
    end process p_mclk;

    -- Test process
    p_test : process is
    begin

        -- input "1001"
        input_1(DATA_IN);
        input_0(DATA_IN);
        input_0(DATA_IN);
        input_1(DATA_IN);

--        -- input "1000"
--        input_1(DATA_IN);
--        input_0(DATA_IN);
--        input_0(DATA_IN);
--        input_0(DATA_IN);

--        -- input "1110"
--        input_1(DATA_IN);
--        input_1(DATA_IN);
--        input_1(DATA_IN);
--        input_0(DATA_IN);
```

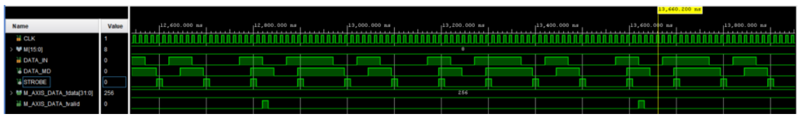
```
        -- Add more input patterns

    end process p_test;

end architecture rtl;
```
Code language: VHDL (vhdl)

In this testbench, we chose a decimation rate of $M = 8$. The output word size is then 9 bits, and the maximum output range is $2^9 - 1 = 511$. We can select different input patterns, and the sinc^3 filter output will be proportional to the number of '1's in the bit stream.

Input pattern = "1001" (50% 1s), output = 256

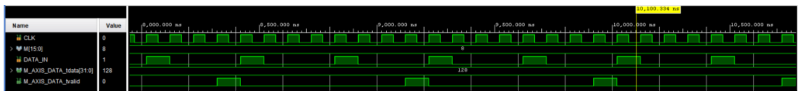


Synchronous decoder

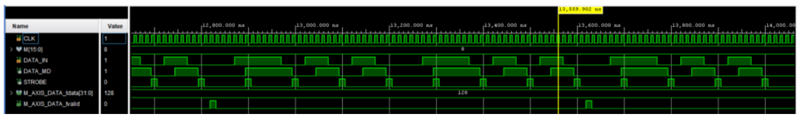


Manchester decoder

Input pattern = "1000" (25% 1s), output = 128



Synchronous decoder

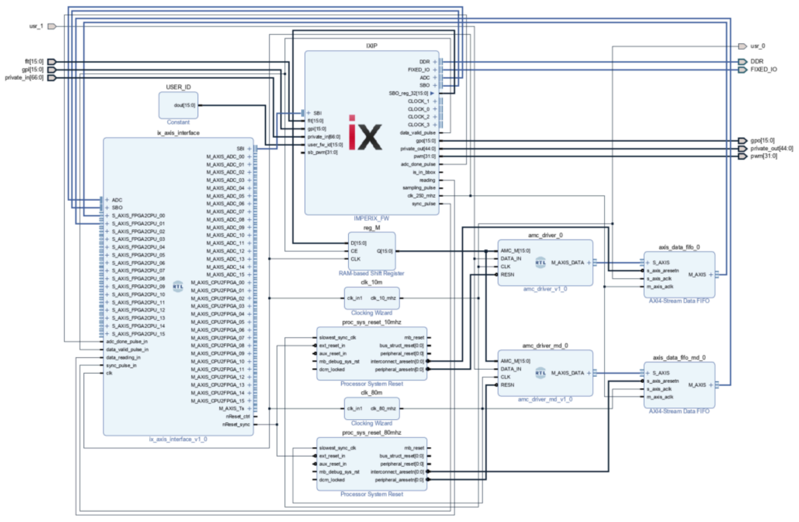


Manchester decoder

Deployment on the B-Board PRO

This Vivado project shows how to add the synchronous decoder and/or Manchester decoder block to the B-Board firmware and send output data to the CPU through `ix_axis_interface`. This project starts from the template introduced in Getting started with FPGA control development, and the following blocks are added to the project.

- `clk_10m` is the 10MHz clock source for the AMC1035 and its output is connected to the physical pin `USR[0]`
- `clk_80m` is the 80MHz clock for Manchester decoding
- `amc_driver_0` and `amc_driver_md_0` are the synchronous decoder and Manchester decoder blocks, their input DATA_IN is connected to the physical pin `USR[1]`
- Two AXI4-Stream FIFOs are used to deal with the asynchronous data transfer between different clock fields, their outputs are sent to the CPU through `FPGA2CPU_00` and `FPGA2CPU_01`
- The decimation ratio M is sent to the FPGA through `SBO_reg_32`
- `proc_sys_reset_10mhz` and `proc_sys_reset_80mhz` provide active low resets for the 10MHz and 80MHz clock fields, their `ext_reset_n` input is connected to `nReset_sync` port of `ix_axis_interface`

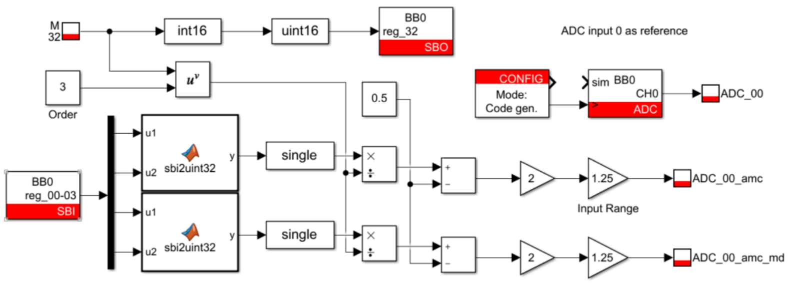


This project has both implementations integrated. In a real application, users can choose one approach or the other, according to their needs.

Before synthesizing the project, Vivado will report timing failure at `reg_M` because the asynchronous data transfer (due to the 250MHz FPGA main clock and the 10MHz/80MHz decoder clock) may lead to a metastable state. However, since we know that, in reality, there will be enough time to wait for a new stable state, this issue can be safely ignored by setting a longer maximum delay. In order to do this, a new constraint file must be added to the Vivado project following the instructions in [Getting started with FPGA control development](). The constraints below shall be sufficient to circumvent this issue.

```
set_max_delay -from [get_pins {top_i/reg_M/U0/i_synth/i_bb_inst/gen_output_regs.output_regs/i_no_async_controls.ou
set_max_delay -from [get_pins {top_i/reg_M/U0/i_synth/i_bb_inst/gen_output_regs.output_regs/i_no_async_controls.ou

set_max_delay -from [get_pins {top_i/reg_M/U0/i_synth/i_bb_inst/gen_output_regs.output_regs/i_no_async_controls.ou
set_max_delay -from [get_pins {top_i/reg_M/U0/i_synth/i_bb_inst/gen_output_regs.output_regs/i_no_async_controls.ou
```

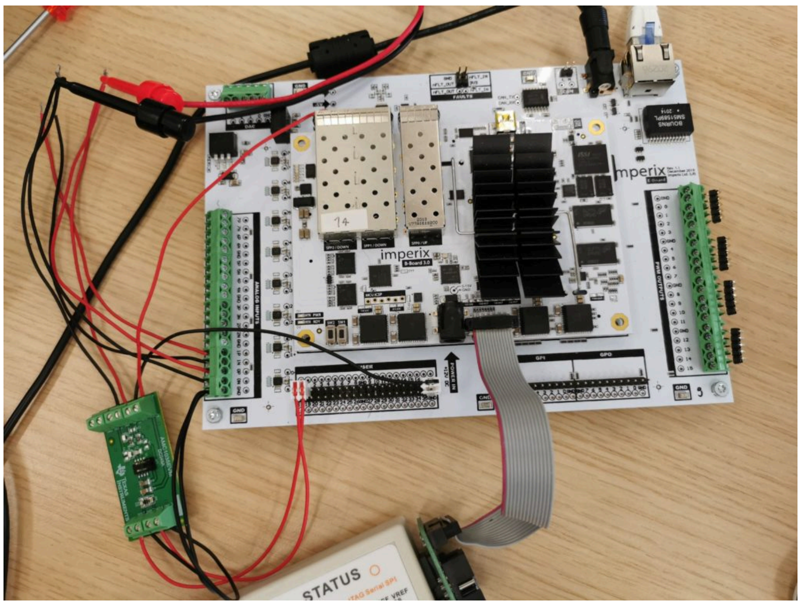
On the CPU side, a Simulink file is provided, which reads the sinc3 filter output and converts the uint32 data to Volts. As introduced in its [datasheet](), the full-scale input range of AMC1035 is +/-1.25 V, then the sinc^3 filter with decimation ratio M converts the input to $1 + 3\log_2 M$ bits. Based on this, data can be recovered using the program below.



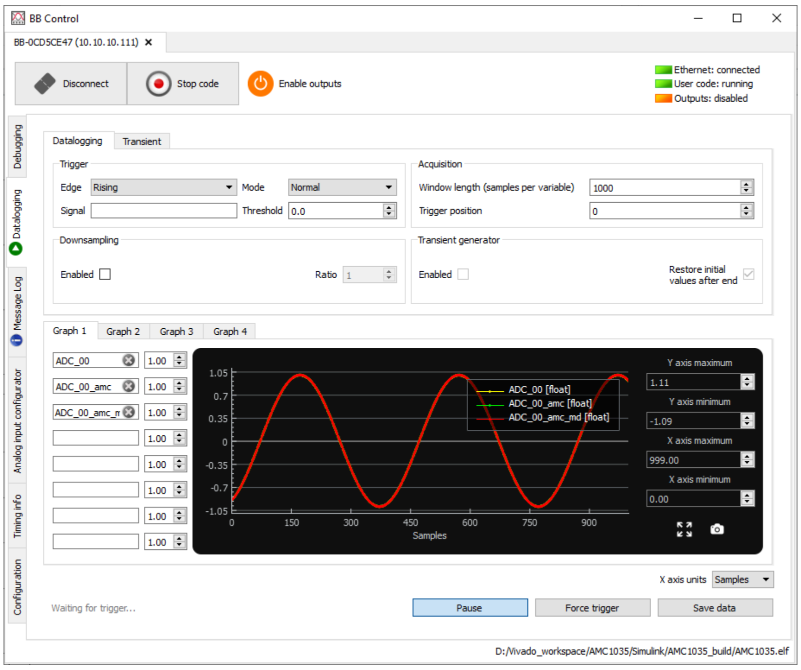
Experimental results

The following hardware was used:

- B-Board evaluation kit
- AMC1035 evaluation module
- Function generator



A 50Hz 2V (p-p voltage) sine wave is connected to ADC 0 of the B-Board and the input port of the AMC1035, and the results are plotted in BB Control. The three signals overlap properly, showing that the synchronous decoder and Manchester decoder blocks for the delta-sigma modulator both work properly.


BB Control

BB-0CD5CE47 (10.10.10.111)

Ethernet: connected
User code: running
Outputs: disabled

Trigger
Edge: Rising
Mode: Normal
Signal:
Threshold: 0.0

Acquisition
Window length (samples per variable): 1000
Trigger position: 0

Downsampling
Enabled ☐
Ratio: 1

Transient generator
Enabled ☐
Restore initial values after end ✓

Graph 1

ADC_00	1.00
ADC_00_amc	1.00
ADC_00_amc_m	1.00
	1.00
	1.00
	1.00
	1.00
	1.00

Y axis maximum: 1.11
Y axis minimum: -1.09
X axis maximum: 999.00
X axis minimum: 0.00

X axis units: Samples

Waiting for trigger...

D:/Vivado_workspace/AMC1035/Simulink/AMC1035_build/AMC1035.elf